CENTRE DE RECHERCHE SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: 05-24146 BIBPOLYTEC
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

CRT-800

A Degree Relaxation Algorithm for the Asymmetric Generalized Traveling Salesman Problem

by
Michel Gendreau[1]
Gilbert Laporte[1]
and
David Simchi-Levi[2]

[1] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, Succursale A, Montréal, Canada H3C 3J7.

[2] Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027-6699, U.S.A.

Centre de recherche sur les transports — Publication #800

September 1991

92-00711/

Dépôt légal – 4ème trimestre 1991

ABSTRACT

The *Generalized Traveling Salesman Problem* (GTSP) consists of determining the shortest Hamiltonian circuit entering and leaving each of m clusters of vertices once and exactly once. This paper describes an exact algorithm for asymmetrical GTSPs in which clusters are pairwise disjoint and each vertex may appear on the optimal circuit exactly once. A lower bound for the problem is obtained from the assignment relaxation of the problem. This bound is then embedded in a tree search procedure. Problems involving up to 400 vertices are solved to optimality.

Keywords: generalized traveling salesman problem, assignment problem.

RÉSUMÉ

Le *Problème du Voyageur de Commerce Généralisé* (PVCG) consiste à déterminer le circuit hamiltonien le plus court entrant et sortant de m ensembles de sommets exactement une fois. Cet article décrit un algorithme exact pour des PVCG asymétriques dans lesquels les ensembles sont disjoints deux à deux, et où chaque sommet ne peut apparaître qu'une seule fois sur le circuit optimal. On obtient une borne inférieure à partir de la solution du problème d'affectation qui constitue une relaxation du PVCG. Cette borne est par la suite imbriquée dans une procédure de recherche arborescente. On résout à l'optimalité des problèmes contenant jusqu'à 400 sommets.

Mots-clés: problème du voyageur de commerce généralisé, problème d'affectation.

1. Introduction

The *Generalized Traveling Salesman Problem* (GTSP) is a well-known extension of the classical *Traveling Salesman Problem* (TSP). Formally, it can be defined as follows. Let $G = (V, A)$ be a graph where $V = \{v_1, ..., v_n\}$ is a vertex set, and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is an arc set. A non-negative cost c_{ij} is associated with every arc (v_i, v_j). The vertex set V is the union of m sets or clusters $S_1, ..., S_m$. The GTSP consists of determining the shortest Hamiltonian circuit entering and leaving each cluster once and exactly once. Several versions of the problem can be defined according to whether the clusters are disjoint or not, and according to the number of times each individual vertex can be traversed. In this paper, we restrict ourselves to the *canonical version* of the GTSP in which all S_i's are pairwise disjoint and every vertex may appear on the optimal circuit at most once. Noon and Bean [12] show how more general classes of GTSPs can be transformed into the canonical form. When $|S_\ell| = 1$ for all ℓ, the GTSP reduces to the TSP.

Several applications of the GTSP have been described in the Operations Research literature. These include the optimal routing of clients through welfare agencies (Saksena [14]), computer files sequencing (Henry-Labordere [4]), post-box location (Labbé and Laporte [6]), and the routing of postal collection vehicles (Laporte, Mercure and Nobert [8]). More recently, Noon and Bean [12] have shown how the multiple TSP with vehicle dependent travel costs can be formulated as a GTSP.

Early solution procedures for the GTSP have relied on dynamic programming (Srivastava *et al.* [15], Saksena [14], Henry-Labordere [4]). Laporte and Nobert [9]

have formulated the symmetric case (defined by $c_{ij} = c_{ji}$ for all $i \neq j$) as an integer linear program and have solved it by means of a constraint relaxation algorithm. Noon and Sepehri [13] have proposed a number of classes of valid inequalities for the symmetric GTSP. In addition, several formulations have been suggested for the general (asymmetric) case. All exploit the assignment problem (AP) structure underlying the problem. They differ in the type of relaxation that is used, and in the structure of the associated branch and bound procedure – see Laporte, Mercure and Nobert [7,8] and Noon and Bean [11].

Recently, Noon and Bean [12] have proposed a clever transformation of the canonical GTSP into an asymmetric TSP. By adequately redefining arc costs, the GTSP is first transformed into a *Clustered Traveling Salesman Problem*, i.e., a TSP in which vertices are partitioned into clusters, each of which must be visited contiguously, and this problem can in turn be easily formulated as a TSP (Chisman [3]). Both the clustered TSP, and the TSP derived from it, contain the same number of vertices as the original GTSP. However, the availability of such a transformation does not imply that it is preferable to solve the GTSP as a TSP. Indeed, while randomly generated asymmetric TSPs can now be solved routinely for several thousands of vertices (see, e.g., Miller and Pekny [10]), the cost matrix obtained from the transformation of the GTSP into a TSP exhibits a peculiar structure which makes the problem more difficult to solve, and it still makes sense to develop specialized algorithms for the GTSP. The object of this paper is to describe a new branch and bound algorithm for the asymmetric GTSP. When applied to randomly generated

problems, it generates a much more parsimonious search tree than any of the known procedures. As a result, larger problems can be solved within relatively small computing times.

The remainder of this paper is organized as follows. In Section 2, we present a mathematical model that has served as a basis for a number of GTSP algorithms, and we extract from it the relaxation used in our algorithm. The branch and bound procedure is detailed in Section 3. This is followed by numerical results in Section 4, and by the conclusion in Section 5.

2. The model

The GTSP can be formulated as a linear integer program as follows. For $i \neq j$, define a binary variable x_{ij} equal to 1 if and only if arc (v_i, v_j) appears in the optimal circuit; for $i = j$, x_{ij} is a binary variable equal to 0 if and only if vertex v_i appears on the optimal circuit. Note that since the solution does not contain any intra-cluster arc, it is not necessary to define x_{ij} if $i \neq j$ and v_i, v_j belong to the same cluster. The problem is then:

$$\text{minimize} \quad \sum_{\substack{v_i, v_j \in V \\ i \neq j}} c_{ij}\, x_{ij} \tag{1}$$

subject to

$$\sum_{v_i \in V} x_{ik} = \sum_{v_j \in V} x_{kj} = 1 \qquad (v_k \in V) \tag{2}$$

$$\sum_{v_i \in S_\ell} x_{ii} = |S_\ell| - 1 \qquad (\ell = 1, \ldots, m) \tag{3}$$

$$\sum_{v_i,v_j \in T} x_{ij} \leq |T| - 1 \quad (T \subseteq V \text{ and } T \cap S_\ell = \emptyset \text{ for at least one but not all } \ell) \tag{4}$$

$$x_{ij} \in \{0,1\} \qquad (v_i, v_j \in V). \tag{5}$$

In this formulation, constraints (2) play a dual role. If $x_{kk} = 1$, meaning that v_k is not used in the solution, then there will be no arc $x_{ik}(i \neq k)$ or $x_{kj}(j \neq k)$ in the solution. If $x_{kk} = 0$, the constraint for v_k is a standard degree constraint that forces one arc into v_k and one arc out of it. Constraints (3) stipulate that exactly one vertex in each cluster must appear in the solution. Constraints (4) are standard subtour elimination constraints: they eliminate all subtours containing vertices of some but not all clusters.

In the approach used by Laporte, Mercure and Nobert [7], constraints (2) are replaced with

$$\sum_{\substack{v_i \in V \\ i \neq k}} x_{ik} = y'_k \qquad (v_k \in V) \tag{6}$$

$$\sum_{\substack{v_j \in V \\ j \neq k}} x_{kj} = y''_k \qquad (v_k \in V) \tag{7}$$

$$y'_k = y''_k \qquad (v_k \in V) \tag{8}$$

$$y'_k, y''_k \in \{0,1\} \qquad (v_k \in V). \tag{9}$$

Relaxing constraints (4) and (8) yields a network flow problem whose solution does not necessarily satisfy the flow conservation conditions. Feasibility is regained by gradually imposing the relaxed constraints through a branch and bound process, in conjunction with Lagrangean relaxation.

In a different paper, Laporte, Mercure and Nobert [8] initially relax constraints (3) and (4). This results in an $n \times n$ assignment problem which may contain illegal subtours, and in which some clusters may be unvisited. These two types of infeasibility are gradually eliminated by branching on the appropriate x_{ij} variables within an enumerative procedure.

Noon and Bean [11] replace constraints (2) with

$$\sum_{\substack{v_i \in V \\ i \neq k}} x_{ik} = \sum_{\substack{v_j \in V \\ j \neq k}} x_{kj} \qquad (v_k \in V), \tag{10}$$

and constraint (3) with

$$\sum_{v_i \notin S_\ell} x_{ik} = \sum_{v_j \notin S_\ell} x_{kj} = 1 \qquad (v_k \in S_\ell;\ \ell = 1, ..., m). \tag{11}$$

Relaxing constraints (4) and (5) yields an $m \times m$ assignment problem in which each cluster is entered and left once, but not necessarily through the same vertex. The authors use this AP relaxation in conjunction with Lagrangean relaxation, within a branch and bound process which implicitly enumerates all feasible vertex subsets. The information provided by the AP relaxation is often sufficient to fathom intermediate subproblems. However, the branching scheme calls for the solution of a full TSP on m vertices whenever the maximum level of the search tree is reached.

We use essentially the same relaxation as Noon and Bean, but an entirely different branch and bound scheme to solve a constrained single m-vertex TSP. This results in much more limited search trees than those obtained with existing algorithms.

3. Algorithm

We have developed a branch and bound procedure for the GTSP, along the lines of the Carpaneto and Toth algorithm for the TSP [2]. Essentially, the algorithm solves a single TSP over the reduced graph $G' = \left(V', A'\right)$, where V' contains one vertex v'_ℓ for every cluster S_ℓ, $A' = \left\{(v'_k, v'_\ell) : v'_k, v'_\ell \in V',\ k \neq \ell\right\}$. With A', is associated a cost matrix $D = (d_{k\ell})$ initially defined by

$$d_{k\ell} = \begin{cases} \infty & (k = \ell) \\ \min\limits_{\substack{v_i \in S_k \\ v_j \in S_\ell}} \{c_{ij}\} & (k \neq \ell). \end{cases} \tag{12}$$

This cost matrix is dynamically updated throughout the search tree and it always underestimates the optimal solution costs. Lower bounds on the cost of the GTSP solution are obtained from the AP solutions associated with D. Any AP solution can easily be interpreted as a choice of m arcs of A. Indeed, if (k, ℓ) belongs to the AP solution, then the arc (v_i, v_j) yielding the value of $d_{k\ell}$ in (12) is selected. Any AP solution may contain two types of infeasibility: (i) illegal subtours on G' and (ii) clusters that are entered and left through two different vertices v_i and $v_{i'}$ of V. Subtours are eliminated by applying the Carpaneto and Toth branching scheme. The second type of infeasibility is dealt with by imposing that either v_i or $v_{i'}$ may not be part of the solution. When this type of branching is executed, the cost matrix D is appropriately updated. An initial upper bound is obtained by

applying a greedy procedure. Throughout the search tree, feasible tours are derived from the AP solutions.

The following notation will be used in the detailed description of the algorithm:

z^* : the cost of the best solution so far identified,

z_h : the value of the AP solution at node h of the search tree,

$\underline{z}_h$: a lower bound on z_h.

I_h : the set of included arcs in subproblem h,

E_h : the set of excluded arcs in subproblem h,

θ_i : the minimum cost of traversing vertex v_i.

ALGORITHM GTSP

Step 0. (Initialization). Obtain a first tour by means of a greedy heuristic: starting from an arbitrary vertex, move to the closest vertex belonging to a cluster not yet visited, and repeat this operation until a tour of cost z^* is formed; this tour is stored. Compute the θ_i's as follows: consider a vertex v_i and its cluster S_p; then $\theta_i = \min_{\substack{v_j \in S_k \\ v_s \in S_\ell}} \{c_{ji} + c_{is}\}$, where $k, \ell \neq p$, and also $k \neq \ell$ whenever $m > 2$. Compute D as in (12).

Step 1. (Node 1 of the search tree). Let $I_1 = E_1 = \emptyset$ and solve the AP with cost matrix D. If $z_1 = z^*$, or if the solution is feasible for the GTSP, stop. Otherwise, execute the patching procedures described in Steps 8 and 9 to obtain a tour of cost $\overline{z}$. If $\overline{z} < z^*$, set $z^* := \overline{z}$ and update the best known tour; if $z_1 = z^*$, stop; if $z_1 < z^*$, insert node 1 in a stack.

Step 2. (Node selection). If the stack is empty, stop. Otherwise select the next node (node h) on which to branch as the node with the smallest value of z_h.

Step 3. (Branching on a subtour). The solution obtained at node h is infeasible for the GTSP. If it contains no illegal subtours, go to Step 4. Otherwise, create descendent subproblems h_r from node h by branching on an illegal subtour. In order to create these subproblems, consider a subtour having the least number t of arcs $\left(v'_{i_1}, v'_{j_1}\right), ..., \left(v'_{i_t}, v'_{j_t}\right)$ of A', in the order in which they appear on the subtour. Then create t subproblems with

$$I_{h_r} = \begin{cases} I_h & (r = 1) \\ I_h \cup \left\{ \left(v'_{i_u}, v'_{j_u}\right) : u = 1, ..., r-1 \right\} & (r = 2, ..., t) \end{cases} \tag{13}$$

$$E_{h_r} = E_h \cup \left\{ \left(v'_{i_r}, v'_{j_r}\right) \right\} \qquad (r = 1, ..., t). \tag{14}$$

In practice, imposing or eliminating an arc is done by modifying the appropriate entries of D. Skip Step 4.

Step 4. (Branching on a cluster). The solution obtained at node h contains at least one cluster entered through some node v_i of V, and left through a different node $v'_{i'}$. A dichotomous branching that eliminates either of these two nodes from the solution is then applied. The "infeasible" cluster S_ℓ selected for branching is determined as follows. We first compute Δ_i and $\Delta_{i'}$ in order to approximate the increase in the objective function resulting from eliminating v_i or $v_{i'}$. The cluster S_ℓ selected from branching is the cluster for which $\min\{\Delta_i, \Delta_{i'}\}$ is maximized. More formally, S_ℓ is determined by

$$\max_{\ell} \min \left\{ \Delta_i = \left(\min_{\substack{j \neq i \\ v_j \in V_h \cap S_\ell}} \theta_j \right) - \theta_i,\ \Delta_{i'} = \left(\min_{\substack{j \neq i' \\ v_j \in V_h \cap S_\ell}} \theta_j \right) - \theta'_i \right\}, \qquad (15)$$

where v_i and $v_{i'}$ are the current entry and exit vertices of S_ℓ, and V_h is the set of vertices of V not incident to an arc of E_h. Creating a subproblem that excludes a vertex is achieved by recomputing the corresponding values of $d_{k\ell}$ as in (12), without considering any excluded vertex in the determination of the minimum.

Execute Steps 5 to 10 for $r = 1, \ldots, t$. Then go to Step 2.

Step 5. (Bounding). Compute a lower bound z_{h_r} on z_{h_r} by row and column reduction of the (updated) matrix D. If $z_{h_r} < z^*$, proceed to Step 6. Otherwise, consider the next r and repeat Step 5.

Step 6. (Subproblem solution). Solve the AP associated with node h_r. If $z_{h_r} \geq z^*$, consider the next r and go to Step 5.

Step 7. (Feasibility check). If the current solution is feasible for the GTSP, set $z^* := z_{h_r}$ and update the best known tour; if $z^* = z_h$, go to Step 2; if $z^* > z_h$, consider the next r and go to Step 5. If the solution is infeasible, go to Step 8.

Step 8. (Subtour patching). If the solution consists of a single tour on G', go to Step 9. Otherwise, apply the following "patching procedure" (Karp [5]).

Define by A'_q the arcs of A' belonging to subtour number q. Repeat the following operations as long as there remain illegal subtours q:

(i) compute

$$\lambda = \min_{q} \min_{(v'_i, v'_j) \in A'_1} \min_{(v'_k, v'_\ell) \in A'_q} \left\{ d_{i\ell} + d_{kj} - d_{ij} - d_{k\ell} \right\}, \tag{16}$$

let q^* and $\left(v'_{i^*}, v'_{j^*}\right), \left(v'_{k^*}, v'_{\ell^*}\right)$ be the subtour and the arcs yielding λ;

(ii) merge subtour q^* with subtour 1, i.e. redefine

$$A'_1 := A'_1 \cup A'_{q^*} \cup \left\{ \left(v'_{i^*}, v'_{\ell^*}\right), \left(v'_{k^*}, v'_{j^*}\right) \right\} \setminus \left\{ \left(v'_{i^*}, v'_{j^*}\right), \left(v'_{k^*}, v'_{\ell^*}\right) \right\}. \tag{17}$$

Step 9. (Tour patching). If the solution consists of a single tour on G, compute its cost $\overline{z}$ and go to Step 10. Otherwise, the solution consists of a single tour $(\ell_1, ..., \ell_m)$ on G', but there exists at least one cluster that is not entered and left through the same node. Let $\overline{S}_u$ be the subset of vertices of S_u visited in the current solution. Repeat the following operation for $u = 1, ..., m$: If $\overline{S}_u$ contains only one vertex v_i, set $i_u := i$. If $\overline{S}_u$ contains 2 vertices v_i and v'_i, compute

$$\rho = \min \left\{ \min_{\substack{v_j \in \overline{S}_{u-1} \\ v_k \in \overline{S}_{u+1}}} c_{ji} + c_{ik}, \min_{\substack{v_j \in \overline{S}_{u-1} \\ v_k \in \overline{S}_{u+1}}} c_{ji'} + c_{i'k} \right\}, \tag{18}$$

where $\overline{S}_0 \equiv \overline{S}_m$ and $\overline{S}_{m+1} \equiv \overline{S}_1$. Let $\tilde{i}$ be the index (i or i') yielding ρ and set $i_u := \tilde{i}$. Let $\overline{z}$ be the cost of tour $(v_{i_1}, ..., v_{i_m})$.

Step 10. (Updating the incumbent). If $\overline{z} < z^*$, set $z^* := \overline{z}$ and update the best known tour; if $z^* := z_{h_r}$, go to Step 2; if $z^* > z_{h_r}$, insert node h_r in the stack, consider the next r and go to Step 5. □

The algorithm was coded in FORTRAN 77 and all assignment problems were solved using the APC FORTRAN code described in Carpaneto, Martello and Toth [1].

4. Computational results

The algorithm was tested on the following three types of randomly generated problems.

Type 1: "Random" problems

The c_{ij}'s are generated in $[0,100]$ according to a uniform discrete distribution. Clusters of equal cardinality n/m are then created by assigning vertices to them in arbitrary fashion.

Type 2: "Euclidean" problems

Here, $2n$ points $P_1, P_2, ..., P_n$, $Q_1, Q_2, ..., Q_n$ are first generated in $[0,100]^2$ according to a continuous uniform distribution. The costs are then defined as

$$c_{ij} = \|P_i - P_j\| \text{ and } c_{ji} = \|Q_i - Q_j\| \tag{19}$$

and rounded to the nearest integer. Clusters are created as in random problems.

Type 3: "Euclidean-clustered" problems

For each cluster, a seed point (a,b) is first generated in $[0,95]^2$ according to a uniform distribution. Then, $2n/m$ points $P_1, P_2, ..., P_{n/m}$, $Q_1, Q_2, ..., Q_{n/m}$ are generated in the square bounded by $[a,b], [a,b+5], [a+5,b], [a+5,b+5]$ and the c_{ij}'s are computed as in Euclidean problems.

For each problem type, for each n and for each m, five different problem instances were generated. All problems were solved on a Silicon Graphics computer, model SGI 4D/280, with a time limit of 200 seconds for each instance. Problems that could not be solved within that time frame were deemed unsuccessful. Computational results are summarized in Table 1: these are average values over the number of successful problems. The meaning of the various table headings is as follows:

TYPE	:	problem type,
n	:	number of vertices in V,
m	:	number of clusters (for each value of n, m is successively taken as $n/10$, $n/4$ and $n/2$),
SUCC	:	number of successful problems out of 5,
AP	:	number of assignment problems solved throughout the branch and bound tree,
STACK	:	number of these problems that could not be fathomed immediately and were inserted in the stack,
SPLITS	:	number of problems that were infeasible because they contained at least one cluster entered and left through a different vertex,
SUBTOUR	:	number of problems that were infeasible because they contained illegal subtours,
TIME	:	CPU time in seconds.

Apart from CPU times, results are rounded to the nearest integer.

These results indicate that the algorithm was quite successful: asymmetric GTSPs containing up to 400 vertices were solved to optimality. This compares favourably

with the algorithms of Laporte, Mercure and Nobert [7, 8] and of Noon and Bean [11], where the size of the largest problems solved did not exceed 100. For a given value of n, problem difficulty varies with the cost structure and with the number of clusters. Random problems are by far the easiest to solve, followed by Euclidean and by Euclidean-clustered problems. (In fact, it is clear that larger random problems could have been solved within the prescribed time limit, provided m had been kept equal to $n/10$). This behaviour is consistent with results observed for other types of routing problems, such as the TSP, for example. For all three types of problems considered, computational difficulty increases with the number of clusters: when m increases, a larger number of illegal subtours are encountered, resulting in more branching; at the same time, the number of infeasibilities occurring because some clusters are entered and left through different vertices decreases with m. However, the net effect of a larger number of clusters is an overall increase in computing time.

A large fraction of the total computing time is spent solving assignment problems. In order to draw comparisons with the algorithms of Laporte, Mercure and Nobert [8] and of Noon and Bean [11], we compare in Table 2 the number and size of APs solved by these two algorithms and by our algorithm. The comparison is limited to Euclidean problems since this is the only case for which comparable data are available. As in the Noon and Bean algorithm, the dimension of our APs is $m \times m$, as opposed to $n \times n$ in the Laporte, Mercure and Nobert algorithm. We also solve significantly fewer APs than in the Noon and Bean algorithm and this explains to a very large extent why our method was so successful.

5. Conclusion

The GTSP is an important TSP extension with applications in a number of practical areas. This paper proposes an exact algorithm for the asymmetric case. As previous algorithms, it is based on the AP relaxation of the problem. As in Noon and Bean [11], it exploits the fact that the optimal solution contains only one vertex in each of one cluster and therefore, only relatively small APs need to be solved. The main comparative advantage of our algorithm lies in its tree search strategy: effectively only one TSP is solved, with appropriate branching to handle infeasibilities. As a result, the number of APs remains very low and thus, relatively large GTSPs can be solved to optimality.

Table 1. Summary of computational results

TYPE	n	m	SUCC	AP	STACK	SPLITS	SUBTOURS	TIME
RANDOM	40	4	5	32	31	26	5	0.13
		10	5	77	68	26	42	0.38
		20	5	29	28	10	17	0.26
	80	8	5	258	238	182	56	1.70
		20	5	392	329	103	227	5.47
		40	5	569	459	61	398	13.19
	120	12	5	1157	1002	549	453	20.14
		30	5	686	570	135	435	15.81
		60	5	1739	1333	217	1116	70.23
	160	16	4	1652	1627	673	955	22.32
		40	4	769	704	210	493	21.47
		80	1	2875	1722	156	1566	100.01
	200	20	5	1644	1464	693	770	29.39
		50	5	11758	1516	405	1110	50.28
		100	2	2349	1895	373	1522	115.59
	240	24	5	599	599	477	122	9.04
		60	5	1783	1769	765	1003	59.04
		120	2	2308	1916	541	1375	122.52

Table 1. (Continued) Summary of computational results

TYPE	n	m	SUCC	AP	STACK	SPLITS	SUBTOURS	TIME
RANDOM	280	28	5	2413	2413	2316	97	45.73
		70	4	1084	1075	653	422	32.60
		140	2	1550	1288	332	956	86.70
	320	32	4	1598	1598	865	733	48.07
		80	5	1421	1421	1234	187	45.96
		160	3	1743	1629	370	1259	134.06
	360	36	4	2169	2169	2128	41	56.62
		90	5	1129	1129	847	282	40.80
		180	1	517	517	246	271	44.31
	400	40	5	2194	2194	2135	58	55.18
		100	2	2978	2978	2748	230	106.81
		200	1	1347	1347	587	760	132.65
EUCLIDEAN	40	4	5	21	19	15	3	0.08
		10	5	37	35	16	18	0.18
		20	5	41	38	13	24	0.36
	80	8	5	143	121	65	56	1.71
		20	5	336	300	84	216	6.00
		40	5	510	429	81	348	15.36
	120	12	5	621	532	188	344	13.54
		30	5	1244	1113	250	862	43.84
		60	4	1274	1078	219	859	68.81
	160	16	5	1300	1186	433	753	43.16
		40	4	1454	1344	357	987	57.90
		80	1	1104	961	122	839	89.56
	200	20	4	1891	1578	463	1117	98.58
		50	1	2090	2022	384	1638	147.01
		100	0	...	...	...	...	...
EUCLIDEAN-CLUSTERED	40	4	5	11	11	10	1	0.05
		10	5	300	215	38	178	1.67
		20	5	6404	3088	12	4168	62.96
	80	8	5	105	78	34	44	1.57
		20	2	1090	750	19	731	21.30
		40	0	...	...	...	...	...
	120	12	3	1081	674	54	644	71.94
		30	0	...	...	...	...	...
		60	0	...	...	...	...	...

Table 2. Number and size of assignment problems solved: comparison of three algorithms on Euclidean problems

		Laporte, Mercure and Nobert [8]		Noon and Bean [11]		Gendreau, Laporte and Simchi-Levi (this paper)	
n	m	AP	SIZE	AP	SIZE	AP	SIZE
40	4					19	4
	9	173	40	887	9		
	10					37	10
	11	567	40	1759	11		
	20					41	20
80	6	11	80				
	8	123	80	4347	8	143	8
	11	145	80	13878	11		
	20					336	20
	40					510	40

Acknowledgements

This work was partially supported by the Canadian Natural Sciences and Engineering Research Council under grants OGP0038816 and OGP0039682. This support is gratefully acknowledged. Thanks are due to Hélène Mercure for her assistance with programming.

References

[1] G. Carpaneto, S. Martello and P. Toth, Algorithms and codes for the assignment problem. *Fortran Codes for Network Optimization*, B. Simeone, P. Toth, G. Gallo, F. Maffioli and S. Pallottino, eds. *Annals of Operations Research* 13 (1990).

[2] G. Carpaneto and P. Toth, Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Science* 26 (1980) 736–743.

[3] J.A. Chisman, The clustered traveling salesman problem. *Computers & Operations Research* 2 (1975) 115–119.

[4] A.L. Henry-Labordere. The record balancing problem: a dynamic programming solution of a generalized travelling salesman problem. *RIRO* B-2 (1969) 43–49.

[5] R.M. Karp, A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM Journal on Computing* 8 (1979) 561–573.

[6] M. Labbé and G. Laporte, Maximizing user convenience and postal service efficiency in post box location. *Belgian Journal of Operations Research, Statistics and Computer Science* 26 (1986) 21–35.

[7] G. Laporte, H. Mercure and Y. Nobert, Finding the shortest Hamiltonian circuit through n clusters: a Lagrangean relaxation approach. *Congressus Numerautium* 48 (1985) 277–290.

[8] G. Laporte, H. Mercure and Y. Nobert, Generalized travelling salesman problem through n sets of nodes: the asymmetrical case, *Discrete Applied Mathematics* 18 (1987) 185–197.

[9] G. Laporte and Y. Nobert, Generalized travelling salesman problem through n sets of nodes: an integer programming approach. *INFOR* 21 (1983) 61–75.

[10] D.L. Miller and J.F. Pekny, Exact solution of large asymmetric traveling salesman problems. *Science* 251 (1991) 754–761.

[11] C.E. Noon and J.C. Bean, A Lagrangean based approach to the asymmetric generalized traveling salesman problem. *Operations Research* 39 (1991) 623–632.

[12] C.E. Noon and J.C. Bean, An efficient transformation of the generalized traveling salesman problem. Working paper, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor (1989).

[13] C.E. Noon and M.M. Sepehri, A cutting plane procedure for the symmetric generalized traveling salesman problem. Presented at the First Triennial Symposium on Transportation Analysis, Montreal and Quebec City (1991).

[14] J.P. Saksena, Mathematical model of scheduling clients through welfare agencies. *CORS Journal* 8 (1970) 185–200.

[15] S.S. Srivastava, S. Kumar, R.C. Garg and P. Sen, Generalized travelling salesman problem through n sets of nodes. *CORS Journal* 7 (1969) 97–101.